Usage of State Space to Control Systems

Finn O'Toole Boire
Programmer
FRC 1678, Davis, CA

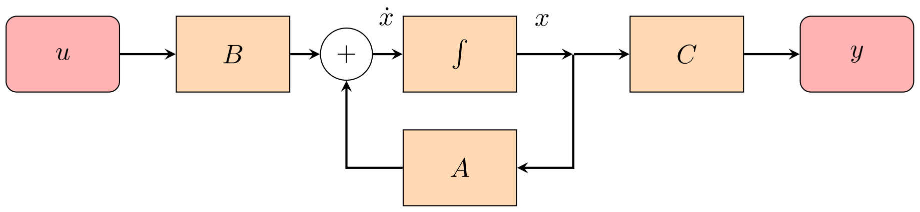
July 22, 2016

Controllers are devices that monitor a system and then alter the state of the system, which generally is some sort of physical changing of the system. In relation to programming our robot, we use controllers to make the various systems on the robot go to desired positions. State space is one of those such controllers, and compared to many other controllers, it is more refined.

1 The Model

1.1 Standard System Model

For state space, you need to have a model of a system. Models can be drawn with a flowchart. This is a block diagram displaying how the input vector u makes the system react, which shown as the vector y.



A, B, and C are all matrices with values and sizes that are determined by the model and number of inputs in the system. A vector, u, contains the output to the system, which generally in robot programmings' case, is in the form of voltage. y is another vector, which contains the sensors that measure the system (i.e. encoders, gyro). There is one more vector as well, x[1], which holds a record of all the states of the system. This is an example x vector.

$$x \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$$

[1] x, u, and y are both functions of time, formally notated $x(t)$, $u(t)$, and $y(t)$. I'm leaving them out to make the equations remain clean-looking

This X vector holds the current angle being held by an arbitrary system and the current angular velocity. If you have other variables that you want to keep track of within your system, like a gyro angle, or encoders, you would add them like so:

$$x \begin{bmatrix} \theta \\ \dot{\theta} \\ X \\ \dot{X} \end{bmatrix}$$

Generally, you don't have to include the second derivative of anything you are keeping in the state vector, because it isn't necessary in the model of the system, as you can see in the flow chart. There is rarely an $\ddot{x}$, only a $\dot{x}$ or x that is ever used as an input or output to any part of the system, as most systems don't pass beyond a second degree differential equation.

1.2 Finding A and B

The A, B, and C matrices comprise of different segments of the model of a system. An easy system to model is that of a motor with a weight, effectively a flywheel, and for such a system, you derive the values of the various matrices with the help of these two formulae:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

To find A and B, you take the model of your system, which you determine by doing math on how the various forces acting on the system affect it. You want to find how the various states of the state vector x, relate to each other and to themselves. This does require some knowledge of matrices, but it's relatively easy.[2] Taking the coefficients of the equations you derive for how each state relates to the others, and you can arrange them in a matrix that lets you multiply your state vector x by A and add the output u to the matrix B to get some sort of change in x, also known as $\dot{x}$. To find C, you need to see how y relates to the states of the system x and the inputs u to the system, much in the same way as you need to see how the states and inputs of x and u relate to the system in finding A and B.

2 State Space Controllers

2.1 Basic Controller

For you to actually affect the system, there has to be some sort of controller that takes in the state of the system, which we have previously defined as x and

[2]See the examples if you want to see how this actually works, because this can seem pretty confusing without something to compare it to.

then outputs something (u) to change the state of the system. In the realm of robot programming, that output is generally voltage.

$$u = -Kx$$

This is the equation that correlates the state of the system to the voltage, where K is a matrix that contains values that you tune for on the system or state space model. The negative sign before K is purely cosmetic and if you leave it out in your controller, it won't affect the end result, from what I understand.

2.2 Finding Poles

To find poles of your system controller, you find the transfer function[3] of your state space model:

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

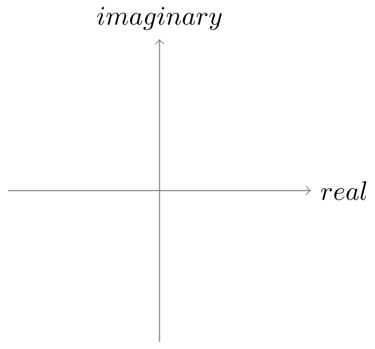
$$\mathcal{L}(\dot{x}) \to sX(s) = AX(s) + BU(s)$$
$$\mathcal{L}(y) \to Y(s) = CX(s) + DU(s)$$

You start by taking the Laplace transform[4] of the model, changing the function to the frequency domain. We want to solve for the ratio of Y(s) to U(s). You can do that by solving the state equation for X(s), or simply using Octave's ss2tf command, which converts a state system to its relative transfer function and makes it easy to find the poles of the function, provided it is a single-input and single-output system. The transfer function typically simplifies to some function that can be represented as a polynomial over a polynomial, like $\frac{Y(s)}{U(s)}$. The poles of any such function are defined as when $U(s) = 0$.

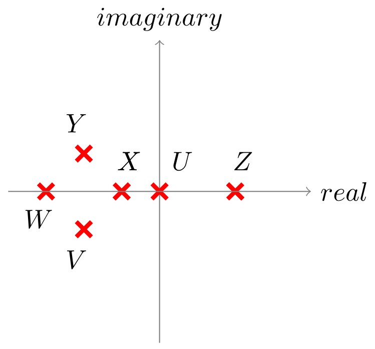
[3]Which is another way to model your system.

[4]Just a little footnote, Laplace transforms are really cool! You multiply by s to take the derivative of a function in the frequency domain.

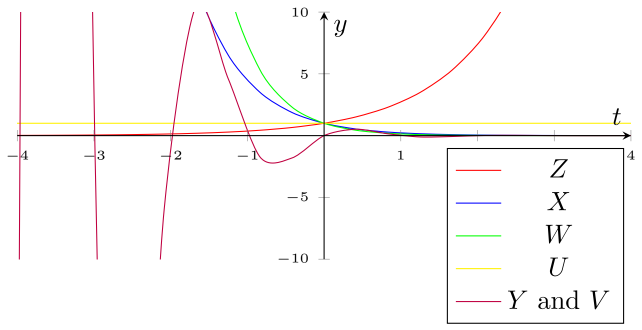
2.3 How Poles Behave



The plane that the polynomial $\frac{Y(s)}{U(s)}$ is represented in is the s-plane, which has a real and complex axis. The poles of the system have real and complex components, which, based off of where they are placed, affect the stability of the system.



The poles that you find with the Laplace transform take the form $y = ce^{at}$ as solutions to the differential equation of x, the state vector. In the equation $y = ce^{at}$, a is the coordinate of the pole, and c is an arbitrary constant. Let's look at what the properties of each of these poles, on this graph, starting with the pole U. With U being at the center of both axes, the Eigen values of the pole are zero, and the plant neither converges or diverges, making it a marginally stable controller and plant. This can also be shown by plugging in the coordinates of the pole. Since a is zero, as t $\to \infty$, it evaluates to simply c, and neither grows infinitely large or small, oscillating indefinitely. Next, Z is in the positive side of the real axis. Positive exponents make systems diverge, and this can be seen if we substitute the coordinates for Z as a. As $t \to \infty$, the solution will grow infinitely large, as any exponent to the positive power will do as it approaches infinity. Now, let's look at X, which has a negative real component and no imaginary component. If you let a equal the coordinates of X, the pole's equation would have a small negative real part, making it converge rather quickly to 0, as when a is negative and t $\to \infty$, the pole's solution will approach 0. Next, looking at W, since it has a greater negative real component, it will also converge on 0, but more quickly, as dictates the properties of exponents. Continuing, if we look at the pole at Y, it has a both imaginary and real part, meaning it is represented as a complex number $(a+bi)$. If we substitute a as a complex number, the solution to the differential equation will be something like $ce^{(a+bi)t}$. Knowing that $e^{ix} = cos(x) + isin(x)$, it shows that any function with an imaginary component will oscillate, and the more of an imaginary component the number has, the faster it will oscillate. Along with that, if there is a pole with an imaginary component, there will also be a pole that is its conjugate. To more easily view how all of these poles affect the output of the K matrix, here is the plot of all of the different poles and how they will make the system converge:



This graph displays the properties of all of the different possible poles and how they change what happens when they are moved about both the imaginary and real plane.

2.4 Finding K

To find K, you need to change the model of the system. Since $u = -Kx$, you can plug that into the state space model, and get:

$$\dot{x} = Ax - BKx$$

Factor out the x

$$\dot{x} = (A - BK)x$$

From here, you can go two different ways, which is to use pole placement or LQR to find the values of the K matrix. We are only going to focus on pole placement, as LQR[5] is much more complicated and I don't have a solid grasp of it yet. However, if you are using pole placement, you are essentially tuning K to make the poles of the system (with the controller implemented) be where you want. By manipulating K, you can move the poles of the system (with controller) so that the solutions oscillate at a slower speed (by making

[5]It allows you to weight your output to the system, so if you are keeping track of multiple inputs and outputs, you can make it prioritize one of the inputs and be more customizeable.

the imaginary component smaller), converge more quickly (by making the real component larger), and etc.

3 Examples of State Space Models

3.1 Mass on a Spring

The model of a system of a mass on the end of a spring with an input force and friction:

$$u + kx + b\dot{x} = m\ddot{x}$$

On this model, we need to define some variables. With this system, you can model it as u being the input force onto the system, as opposed to voltage like we will do on the flywheel. Along with that, k is the spring constant, and x is the displacement of the spring. Friction is modeled as a constant b multiplied by your velocity, or $\dot{x}$. And finally, m is mass, which you are multiplying by $\ddot{x}$ to get a force. So in total, your system is the input force u - the spring force kx - the friction force $b\dot{x}$ equals the total force, or $m\ddot{x}$. Now, having established the model of the system, we can begin solving for the A, B and C matrices. If we begin by re-naming the variables, we can change $x = x_1$, $\dot{x} = x_2$, and $u = u_1$. Notice that there is now a relationship between x_1 and x_2, as $x_2 = \dot{x_1}$. Now, if we re-write the model to be based off of these new variables, it will look like this:

$$u_1 - kx_1 + bx_2 = m\dot{x_2}$$

Now, what we want to do is take each of the states and look at its dynamics, meaning, finding the derivative of each of the states in terms of the other terms, x_1, x_2, and u_1.

$$\dot{x_1} = x_2$$
$$\dot{x_2} = \frac{-k}{m}x_1 + \frac{-b}{m}x_2 + \frac{1}{m}u_1$$

We want to still know how $\dot{x_1}$ relates to itself, x_2, and u_1.

$$\dot{x_1} = 0x_1 + 1x_2 + 0u_1$$
$$\dot{x_2} = \frac{-k}{m}x_1 + \frac{-b}{m}x_2 + \frac{1}{m}u_1$$

And knowing this, that the vector of states $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and the vector of inputs $u = \begin{bmatrix} u_1 \end{bmatrix}$, then we can find the matrices A and B, because they are multiplied as Ax and Bu to get $\dot{x}$. This means that the matrices

$$A = \begin{bmatrix} 0 & 1 \\ \frac{-k}{m} & \frac{-b}{m} \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}$$

And by doing the same process on y, which equals x_1, you can find the C and D matrices.

$$\dot{y} = 1x_1 + 0x_2 + 0u_1$$

$$C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 \end{bmatrix}$$

After you find all the matrices, you have a state space model of the system. To tune the controller, whose equation is $u = -Kx$, you begin using pole placement or LQR, and you don't need a physical system to test it on, as the state space model allows you fairly realistic system response for the tuning of K.

3.2 Flywheel

Knowing the model of a motor with relation to voltage and angular velocity is

$$\ddot{\theta} = \frac{K_T}{JGR} V - \frac{K_T K_V}{JG^2 R} \dot{\theta}$$

where K_T is the coefficient of torque, J is the moment of inertia of the system, G is the gear ratio, K_V is the voltage constant (related to backemf), and R is the resistance of the motor. and we assume that our state vector and input vector

$$x = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$$

$$u = \begin{bmatrix} V \end{bmatrix}$$

We can solve for the values of A and B by writing the model in terms of x_1 and x_2.

$$x_1 = \theta$$

$$x_2 = \dot{theta} = \dot{x_1}$$

$$u_1 = V$$

Substitute it into the model

$$\dot{x_2} = \frac{-K_T K_V}{JG^2 R} x_2 + \frac{K_T}{JGR} u_1$$

Notice that x_1 doesn't show up in the model, so now, when we take the derivative of x to find how the states relate to each other, it is equal to zero.

$$\dot{x_1} = 0x_1 + 0x_2 + 0u_1$$

$$\dot{x_2} = 0x_1 + \frac{-K_T K_V}{JG^2 R} x_2 + \frac{K_T}{JGR} u_1$$

To find A and B, you take the components that multiply together to form the resultant vector $\dot{x}$ (Ax and Bu), and plug them in.

$$A = \begin{bmatrix} 0 & 0 \\ 0 & \frac{-K_T K_V}{JG^2 R} \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ \frac{K_T}{JGR} \end{bmatrix}$$

Then, to find C and D, you do the same for y as we did in the model of the spring, where K_c is the conversion constant.

$$y = K_c x_1 + 0x_2 + 0u_1$$

$$C = \begin{bmatrix} K_c & 0 \end{bmatrix}$$

From there, you have a working model that you can then use pole placement to find the values of K, and tune your controller off of.

4 Conclusion

I hope this documentation on how state space works has been helpful, and there is still very much to add. I'm going to be updating this as I learn more about state space, but for now, I hope this will help anyone who is looking to both model and create a controller with a state space system. Along with this document, I'd like to link[6] this YouTube channel, which has helped me understand a lot of the concepts that I talked about and explained in the document.

[6]https://www.youtube.com/user/katkimshow/videos